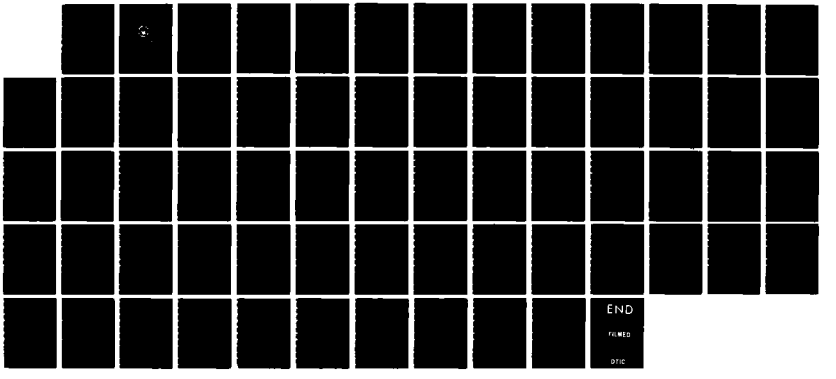
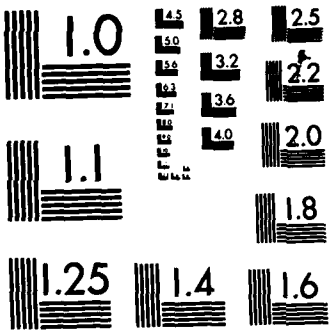
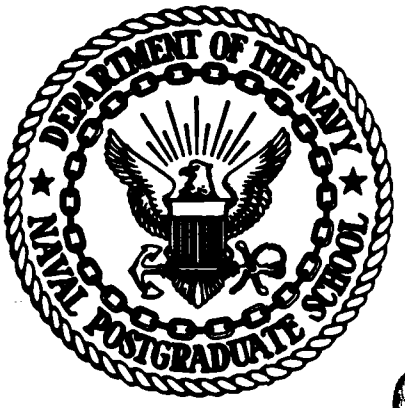
AD-A159 813 REUSABLE SOFTWARE: TRADE-OFF ANALYSIS AND A NEW APPROACH(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA C A MURNAN JUN 85 1/1

UNCLASSIFIED F/G 9/2 NL





NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

REUSABLE SOFTWARE:
TRADE-OFF ANALYSIS AND A NEW APPROACH

by

Cynthia A. Murnan

June 1985

Thesis Advisor: Gordon H. Bradley

Approved for public release; distribution is unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A159 813	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Reusable Software: Trade-off Analysis and a New Approach		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Cynthia A. Murnan		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		12. REPORT DATE June 1985
		13. NUMBER OF PAGES 63
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

reusable software, reusability, software engineering

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Software reusability is seen as a resource which can assist in resolving the current software crisis. This thesis discusses issues that are relevant to the concept of reusable software. It reviews the definition of reusable software and presents software development scenarios to describe possible guidelines for performing a trade-off analysis in determining the pros and cons of incorporating reusable software concepts (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (Continued)

into a software product. The thesis also suggests the need for new and dramatically different methodologies to make software reusability a viable concept.

Approved for public release; distribution unlimited.

Reusable Software: Trade-Off Analysis and A New Approach

by

Cynthia A. Murnan
Lieutenant, United States Navy
B.S., State University Of New York, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1985

Author: Cynthia A. Murnan
Cynthia A. Murnan

Approved by:
Gordon H. Bradley, Thesis Advisor

Bruce J. MacLennan, Second Reader

Bruce J. MacLennan, Chairman,
Department of Computer Science

Kneale T. Marshall, Dean of
Information and Policy Sciences

ABSTRACT

Software reusability is seen as a resource which can assist in resolving the current software crisis. This thesis discusses issues that are relevant to the concept of reusable software. It reviews the definition of reusable software and presents software development scenarios to describe possible guidelines for performing a trade-off analysis in determining the pros and cons of incorporating reusable software concepts into a software product. The thesis also suggests the need for new and dramatically different methodologies to make software reusability a viable concept.

Additional keywords: life cycle models; trade-off analysis; software engineering; scenarios.

TABLE OF CONTENTS

I. INTRODUCTION

A. DEFINITION OF REUSABLE SOFTWARE

There has been increasing attention given to the concept of "reusable software". The Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management held a software workshop in June, 1981 which contained a panel devoted to an evaluation of reusability [Ref. 1]. ITT Programming sponsored a workshop on reusability in programming in September, 1983 [Ref. 2]. Papers from this workshop were incorporated into the September, 1984 IEEE Transactions on Software Engineering [Ref. 3].

A clear definition of what is meant by "reusable software" is necessary if we are to pursue the value of reusable software as a resource in the development of software products. The JLC report of the panel on software reusability defined reusable software as "...existing software, including specification, design, code, and/or documentation, which can be employed or adapted, in part or total, into a new end use." [Ref. 4] The important point to notice in this definition is that reusable software is seen as encompassing any information produced throughout the life cycle of the software product, from specification to design, from coding to maintenance. It must be realized that software reusability is not just relevant for program

code generation and, in fact, is possibly least concerned with that aspect as an element for reusability.

The definition does not include reuse of software by multiple users on multiple occasions, such as the use of an operating system or a compiler on several different types of processors. The definition of reusable software given above focuses on software for which the end-use is totally, or in part, new.

B. MOTIVATING FACTORS FOR REUSABILITY

The primary motivating factor for researching the viability of reusability in software development is the current software crisis. Presently, the development of computer systems is delayed by the development of the software for those systems. The production of the software is very labor intensive. There are no standard, reusable components, such as can be found for the hardware of the system. The software components are basically developed from scratch. Software products are also continuing to grow, both in size and complexity. There are not, however, enough software engineers available to deal with these increases without added delays in the development of software applications. It is felt that reusable software could assist in alleviating these problems.

Another factor involved, especially for the Department of Defense, focuses on embedded computer systems. VLSI

technology and microcomputers have pushed for "...complex embedded software implementing specific functions in these small hardware elements." [Ref. 5] If the software can not be reused, the hardware will not be easily extendable for different applications, and the technological gains for system development will be lost in the development of the software.

An additional motivating factor is that reusable components may be more reliable. More effort must be placed on testing and documenting components that will be reused. Also, maintenance will be performed on these software elements at each location that chooses to use them. Thus, if they have not undergone many changes to meet the various applications, repeated maintenance will be performed, with any pertinent information from one site being available to others that may reuse those same components.

C. SCOPE OF THESIS

We must recognize that, at the present time, the use of reusable software may simply not be practical in every situation. A software engineer must be able to ascertain and study the pros and cons, and perform a trade-off analysis to make the best possible decisions regarding software reusability. An effective trade-off analysis must concentrate on the benefits, if any, which can be obtained by incorporating reusable components. The development

manager must consider all the issues of reusability and trade off the possible economic savings of using previously devised software components with the possible economic costs of trying to locate such reusable components. If reusable components are not available, it may not be cost effective to expend additional time and effort to extract the information from manuals, listings, and other documents. If the product being developed is completely new in all regards, reusable components may not yet exist. A software manager must be able to consider the entire project itself and trade off the possible benefits of reusability with any increased man-hours spent on the project in attempting to use reusable components, which may, in turn, result in increased costs and schedule delays. Additionally, the extra cost of developing *reusable* components that may then be used on later projects must be considered. Designing components for later reuse creates a need for more generalization, more documentation, and more thorough testing. For the initial projects within a development office to incorporate reusable software components, increases in time, effort, and initial cost outlay will occur. It will take foresight and enlightenment to be able to see beyond one's own individual project to the possible benefits for many projects.

It is, therefore, important that the software engineer or software development manager have some guidelines with

which to make an informed decision. The entire life cycle of the software system to be developed must be considered for reusability. A life cycle model, as described by Barry Boehm, is depicted in Figure 1 [Ref. 6]. It presents the various phases involved in the development of a software product. The software manager must consider reusability for each of these phases: reusability of requirements/specifications, reusability of preliminary and detailed design information, reusability of code, and reusability of maintenance information. Reuse of test and integration plans must also be considered. This thesis presents software development scenarios to delineate the areas which must be studied in a trade-off analysis. These will then project some guidelines which the software development manager can consider when faced with the issue of reusable software for the project at hand.

Beyond the issue of determining the pros and cons of reusability for systems currently being developed, we must consider the future of software reusability, and what can be done to make it a more practical reality. The initial concept of reusability can be traced back to 1968 [Ref. 7]. Since that time, much discussion on the topic has followed, with much agreement as to the positive value of reusability. However, few systems have been developed by incorporating reusable software. In addition, new systems have not been developed to better allow for future software reusability.

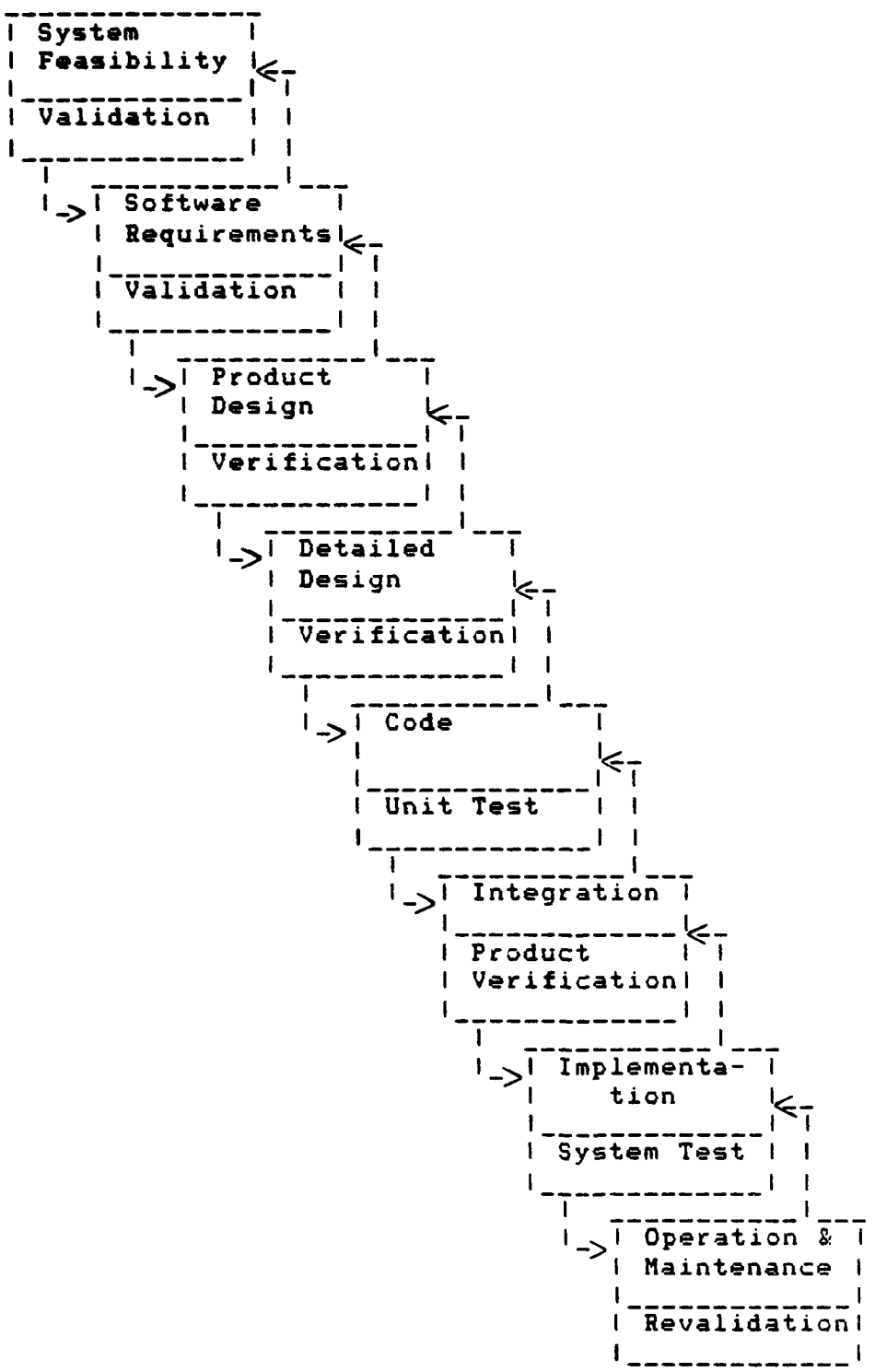


Figure 1. Life Cycle Model

A possible solution to this problem of how to get reusability started on a wide-scale would be to create a project specifically designed for this purpose. This project would be tasked with determining what components could be reused, how to write the information down in a standard, understandable, easily retrievable form, and producing a cataloging system for DOD-wide usage. By being specifically assigned as a project in its own right, we avoid the problem of determining who will assume the burden of the initial cost investment. If the Department of Defense is to get software development under control, the importance of reusability must be comprehended and assured.

The notion of a separate project to develop standard, reusable software components may be viewed as being analagous to the Navy's development of standard shipboard computers. The Navy has pursued, as a specific task, a standardization program for tactical computers to decrease problems of maintenance and logistics [Ref. 19]. The AN/UYK-7 was developed by UNIVAC as a general purpose standard for shipboard and shore use. It was designed for maximum modularity to allow adaption to various applications [Ref. 20]. Thus, the design of this standard computer incorporated modularity which, in effect, allowed reuse of standard parts for different end uses. This standardization of Navy computers was intended to "...eliminate intersystem incompatibilities, complex

B. POSSIBLE SOLUTION

The question now becomes, "How do we invest the initial capital and thereby get reusability rolling?" We have seen throughout history that the method of increasing productivity involved standard components and the capability of mass production. We must be able to incorporate these same ideas into software development. To generate software components, including code or other development information, that will be flexible enough to be used in many applications, we must be able to look beyond one project to the software industry in total. We must expend capital at the outset to generate reusable components and a viable reusability methodology to be able to avoid the duplication of effort and the basic inefficiencies of having a single person or group develop a software product from scratch, producing one product at a time.

Our present practices do not lend themselves easily to incorporating reusability. We generally emphasize cost reduction and product optimization. A software manager is basically only concerned with the development of the system for which he is responsible and accountable. He is unlikely to strive to create a flexible product or one that employs new techniques which will assist in future reusability as this would undoubtedly result in additional costs, effort, and time spent on his project.

is because, at the present time, we resort to measures such as increasing the number of personnel involved on a project in the hopes of increasing the production rate, which, in turn, increases costs. We must be willing to learn from history and attempt to determine whether past successful techniques can apply, or be modified to apply, to the current software production problems.

If we are to attempt to resolve the software crisis, we must view software development as being capital-intensive. According to Wegner, "A production process is capital-intensive if it involves expenditures early in its life cycle for the purpose of increasing productivity later in the life cycle." [Ref. 17] Thus, a capital-intensive view of software development is vital to the issue of reusabilty. As Wegner further noted, "The long term objective of capital-intensive software technology is a systematic process of capital formation that provides a stock of tools from which application programs can be cheaply and reliably constructed, and allows flexible enhancement in response to changing requirements and replacement in response to changes in the technology." [Ref. 18] By investing capital up-front, we will be able to design software components that will allow for reusability in the future. Flexibility will be enhanced as components will be able to be used in producing different end-products.

III. GETTING REUSABILITY STARTED

A. INITIAL CAPITAL INVESTMENT

Much of what we are currently experiencing in the production of software today was confronted in the 18th century by early manufacturers. These early craftsmen discovered that the problems created by their expanding industries, such as how to find enough qualified personnel, how to improve methods to keep costs down and increase performance, and how to increase production while decreasing costs, could be resolved by mass production techniques. Thus, we find that they invested capital to create mass production tools and to develop standards, which, in turn, led to the appearance of interchangeable parts. This initial cost outlay was recovered in the increased productivity which resulted. Products, such as clocks, which were initially handmade by a single skilled craftsman, one at a time, could, with the advent of mass production techniques, be produced much more quickly, and at a reduced cost [Ref. 16].

The software industry currently faces some of these same issues. There is a shortage of qualified software engineers and programmers. It is difficult to find methods which will decrease costs. In fact, software costs continue to rise dramatically. Also, it is difficult to increase the production rate and yet maintain or decrease costs. This

extensive training and experience to thoroughly understand. Much of the present emphasis on reusability is focused on reuse of code, since this can be encouraged without first having to resolve all of the above listed problem areas. Code is developed using a modular approach. This can be very effective for similar types of projects, and should be actively pursued. As Horowitz and Munson related, "Input and output routines, report generating routines, computational and processing routines are all designed and written by the staff of analysts and programmers on the project. This is clearly an unfortunate situation as much of the code of one system is virtually identical to code which was previously written." [Ref. 15] This is a situation that we can and should, at least, be working toward resolving.

The economic value of reusability beyond just reuse of code warrants a concerted effort to resolve the above problems. The software development process must incorporate measures to include reusability, and the software managers and engineers must necessarily involve themselves in this issue if we are to ever alleviate the ever-expanding software crisis.

with a system such as SADT which requires extensive training to comprehend.

B. CURRENT DEVELOPMENT CONCEPTS / PROBLEMS

Current practices for the development of a software product involve very little consideration for reusability. Most people will agree that the term "reusable software" presents a very appealing notion, however, the reality and practicality of it presently do not exist.

There are several problems which create a hindrance to the use of reusability. As Horowitz and Munson pointed out [Ref. 14], in order for reusable software to become a reality, we must be able to determine which components are capable of general usage and of being specified so that they can be used by others on different projects. We must also be able to write the information in a descriptive form that is readily understandable. The method of cataloging these reusable components must also be delineated, so that others will be able to discern what information is available for reuse.

Inasmuch as we have not yet addressed these problems on a wide scale, we cannot hope to see the reality of reusability on anything but a limited basis. Currently, in a development project, most of the information generated is recorded on masses of paper, in manuals, using a locally-developed methodology, or using a system that requires

Software products are changed when the current programs, after going operational, are found to be inappropriate, incorrect or ineffective. Problems corrected during the maintenance phase could have originated anywhere in the life cycle. Those created earliest are the most costly to fix during the maintenance phase. Thus, if the maintenance information for one product could be reused perhaps similar problem areas could be avoided earlier on in the development process, resulting in great overall savings.

As the above discussion relates, there is much that can be reused from each of the various life cycle phases in the development of a software product. We should not limit ourselves to reuse of code alone. We need to recognize that we must be able to reuse ideas and concepts and thought processes. This information must be made readily available so that we can avoid previous mistakes, duplication of effort, or simply time consuming determination of specifications and designs.

This information must also be easily retrievable in a form that is readily understandable. If it is not, people will find it more advantageous to duplicate the effort than to spend the time and energy necessary to discern the information from some source. Often, data is retained from the various processes of the development of a product. However, it may be written in a voluminous document or

decisions and difficulties encountered during this phase could prove useful to future software development projects.

7. Implementation Phase

It is in this phase that the fully functional system is assembled and tested to ensure that it meets the requirements. The information generated during the implementation phase could show others the validity of different aspects of the system and thereby assist in the initial requirements phase of other products.

8. Maintenance Phase

The maintenance phase includes everything concerning the software product that is done to it after it becomes operational. This includes error correction and modifications. This phase is important because so much of the cost of generating a software product is consumed by the maintenance phase. Figure 2 shows that, on the average, about 40% of the overall hardware-software dollar goes toward maintenance. This also represents an average of approximately 70% of the overall cost of the software [Ref. 13]. The information generated during this phase should be well documented and readily available for reuse. Maintenance plans used on one project may be applicable to another. Errors or inadequacies encountered during this phase could be considered and accounted for in creating new products.

testing is carried out effectively and efficiently. These plans could prove to be reusable for the testing of systems which may be similar in some regards to the system for which the plans were originally devised.

5. Coding Phase

It is during the coding phase that the software product is actually implemented in a specific programming language. If the code is developed using recognizably useful techniques, such as modularity and abstraction, the code may be able to be reused. It is this aspect of reusability that R. Lanergan and his associates had so much success with. They felt that "...there are some business functions which are universal routines and others which are common to a company or functional area. These routines, can, therefore, be prewritten." [Ref. 12] They produced logic structures, prewritten for each of six types of business application programs, such as update, select/edit, and report, and found that they could reduce 40-60% redundancy in business application development. This type of reusability has potential for locally-applicable usage at the present time with little more than a directive to do so, combined with the management and effort to make it work.

6. Integration Phase

It is in the integration phase that the individual components are combined into the final product. The

Thus, it can be seen that techniques presently available for recording requirements analysis information are insufficient for increasing the use of reusability. What is needed is a more universal, more understandable method for saving this information.

3. Product Design Phase

The product design phase defines the overall hardware-software architecture, the parts of the system and their relationships, the basic algorithms that will be used, and major data representations. There is not much detail generated in this phase, but the general information developed could be vital for producing other software products. The ideas that lead to the determination of the information used in this overall product design phase should be recorded so that following projects will be able to benefit from the work of others and duplication of efforts can be avoided.

4. Detailed Design Phase

Greater detail is generated during the detailed design phase. Precise algorithms, data structures, control structures, and interfaces are developed. This design information should be retained for reusability purposes. This phase leads into the coding phase so the information could be useful for coding in various different languages.

Also developed during the design process are test plans for the system, which will be used to ensure that

software product is generated. Much discussion and effort goes on in this phase to ensure the best possible description of what the system is to do. The pros and cons of various requirements are discussed as decisions are made. This information should be recorded and readily available for reusability, as many of these same initial questions are asked over and over again in the development of other software products.

There have been efforts to present systems which capture this requirement information, however, these systems are not universal or wide-spread in use. One such system is the Software Requirements Engineering Methodology (SREM) [Ref. 10]. It was developed for real-time systems by TRW and is driven off a centralized relational database. SREM is a message oriented system that uses a stimulus-response model of real-time systems. SREM is a viable system, but it is not cost effective for small and medium sized projects. Extensive training is also required to effectively use it.

Another methodolgy for requirements analysis and definition is the Structure Analysis and Design Technique (SADT) [Ref. 11]. This technique uses diagrams to model processes or data. There are problems with this system also. The models built in SADT are often difficult to immediately transform into designs. The diagrams are also complex and difficult to understand, requiring much training to become adept at using them.

II. SOFTWARE REUSABILITY LIFE CYCLE CONCEPTS

A. REUSABILITY THROUGHOUT THE LIFE CYCLE PROCESS

In keeping with the previously discussed definition of reusable software, which encompasses all aspects of the software product, the various life cycle phases should be examined to determine their potential for contributing to reusability. (See Figure 1 for diagram of life cycle phases).

1. Feasibility Phase

The first phase in the life cycle model of the development of a software product is the feasibility phase. It is concerned with defining a basic approach for the entire development project and determining the feasibility of that approach throughout the life cycle. Much of the information discerned in reviewing various approaches could prove to be very useful to the development of other software products. If the information could be effectively saved and readily retrieved, it could be reused in other projects. The superiority of one approach to another would be recorded, along with the reasoning behind this determination. In this way, the information would not have to be reproduced and much effort would be saved.

2. Requirements Phase

During the requirements phase, the specification of the required functions, interfaces, and performance of the

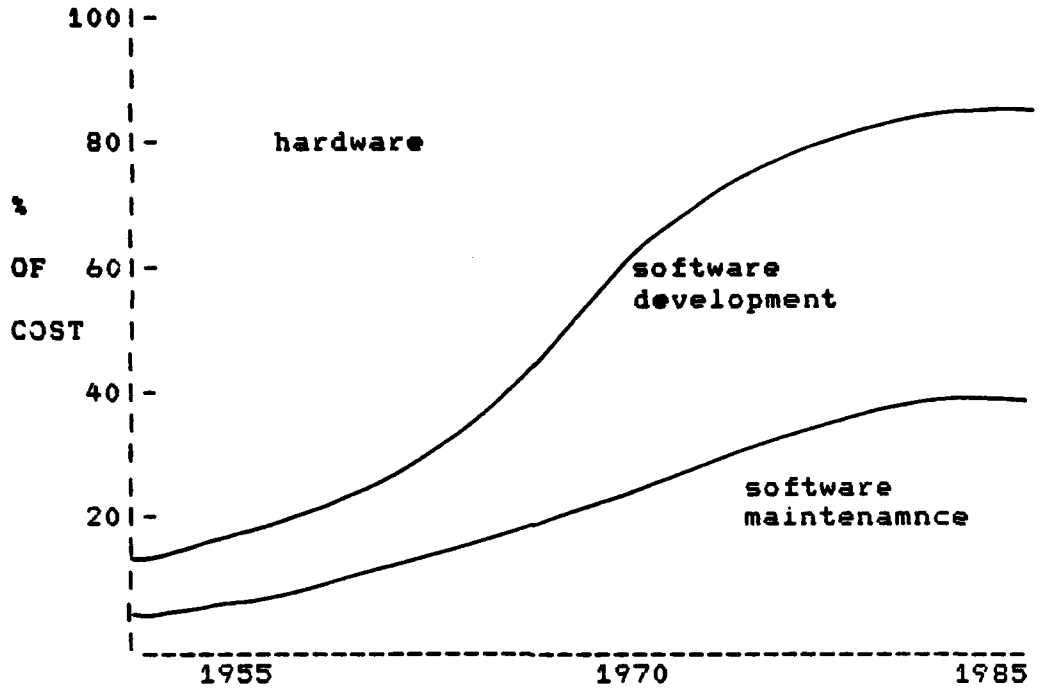


Figure 2. Hardware-Software Cost Trends

The need for reusable software exists, the reality presently does not.

There have been limited success stories. Robert Lanergan, manager of advanced software development at Raytheon Company's Missile Systems Division, announced in 1979 successful use of reusability [Ref. 8]. That effort, though, centered around reusable code for products performing very similar functions. It did demonstrate, however, that reusability is effective, at least on a limited basis.

A greater need exists, though, to expand this capability. DOD software costs continue to grow extensively. Figure 2 shows the trend for the increasing software percentage of total system costs [Ref. 9]. We must be able to develop reusable software, and a corresponding support system, so that reusable components will be available to personnel throughout the Department of Defense, not just in one development shop. This thesis suggests the need to recognize that new and dramatically different methodologies must be devised if we are to see the development of reusable software on a large scale. Otherwise, we will have to be content to incorporate reusable software on a more or less "local" level, that is, basically code, within one relatively small development shop, for similar functions.

personnel and logistic support problems, and uneconomical procurement programs." [Ref. 21] It can be noted that these same problems exist today for software systems.

The standardization program for non-tactical computers currently involves replacing some older systems with more advanced ones. For example, the AN/UYK-5 computers are being replaced with AN/UYK-65 computers. This, too, is being accomplished via a specific program. Additionally, another project has been tasked with supplying computers to ships that do not presently have non-tactical computers.

Thus, it can be seen that the idea of a specific project to handle such a broad scope is not new. It may be that this approach presents the only logical plan for getting reusability started on a wide scale.

Another point to notice is that more mature industries have increased productivity by a division of labor, or specialization. It may be worthwhile to ponder the benefits of doing the same in the software industry. We might develop software specialists in the fields of specifications, designs, maintenance, or code production. These specialists could be involved in generating the initial reusable components from existing software systems within a specified project, as described above. Additionally, personnel might perform specialized tasks on individual software development projects.

It has been obvious from the past and present lack of the widespread adoption of reusability that current practices toward this end are insufficient. Thus, it becomes imperative to contemplate alternative measures. These must look beyond traditional methodologies to more inventive means of reaching the primary goal: resolving the software crisis.

IV. TRADE-OFF ANALYSIS

A. BACKGROUND

Using current practices, it is not feasible to achieve reusability on every software development project. This includes both trying to incorporate previously developed software components into the project at hand and attempting to develop the software components of the present project with future reusability in mind. The software manager must perform a trade-off analysis to determine if reusability will be economically justified. To accomplish this, he must first know what factors should be considered. Presently, there are no guidelines set forth which would assist in making the necessary trade-off decisions. Thus, reusability is often employed in a project today only when an effort is made to expressly direct personnel to do so. The result is that most development projects continue to be basically started from scratch, and any economic gain which may be obtained by using reusable components is lost.

A trade-off analysis must focus on the costs and benefits to be attained on a software development project by having reusability. For reusability to be appropriate, the benefits must be balanced against the costs of developing the system with reusable components.

This thesis now presents various scenarios designed to offer insight into the important issues of a trade-off

analysis. These scenarios are used to represent different development situations which define varying degrees of the ability to incorporate reusability. Accordingly, the types of analysis which must occur to determine whether reusability is practical for a particular project are depicted. The outcome of the presentation is an outline of possible guidelines to use in evaluating the reusability trade-off decisions.

B. REUSABLE COMPONENTS

Before introducing the scenarios, it is worthwhile to reconsider exactly what software components have the potential for reuse. These should be kept in mind, as the scenarios presented below will attempt to show the appropriateness of reusing certain components in various different situations.

One aspect of a software project that has potential for reuse is the information that has been generated during the feasiblity phase of past projects. This information is useful in determining an overall approach to a project, and can assist in accelerating the start of the succeeding phases in the development of a new product.

Next, the requirements and specifications of previous projects may be determined to be appropriate for reuse. Design information as well as test plans for the system may

be available. Also, the code itself may be suitable for reuse, including subroutines, functions, and modules.

Information produced as the result of the operation of a previous system may prove useful to a new system. Maintenance data generated as past systems are supported may also provide insight and guidance for a new project.

In general, any information that can be documented in an understandable and retrievable manner has the potential for reuse in a totally, or partially, new end product. Any new project that can reuse some, or all, of the above components, from past projects may benefit by a decrease in system development time and effort, decreased development costs, and/or the production of a more effective system.

C. DEVELOPMENT SCENARIOS

Development scenarios are now presented to better illustrate the trade-off analysis necessary to determine whether reusability for a project is appropriate.

1. Scenario I.

Command Alpha has just received computer graphics hardware to increase their capabilities. They desire the graphics facility to be able to produce technical drawings for the command. The computer came with some software, but did not come equipped with a figure generation package. The development office of the command is therefore tasked with producing an interactive figure generation system to meet

the needs of the command. A software development team is formed to accomplish the task.

Since the team is part of the command which will use the finished product, they can be viewed as end users of the system themselves. Thus, they sit down to determine how to proceed, what the requirements of the system should be, and so forth.

The initial deliberations center on the basic approach to be taken for the entire project. The team leader discovers that little has been documented in the past as to how other groups have determined what the best approach is in producing a figure generation capability. Therefore, much discussion on this topic ensues. The team leader appoints one member of the team to record all considerations and conclusions made as the project progresses. Different approaches are analyzed, with the group finally deciding to proceed with a menu-driven system, with selection provided via mouse controls.

At this point, the team leader must determine whether to proceed on this project with future reusability in mind. That is, he must decide if it will be beneficial to expend more money now to achieve savings later. He considers the following:

- Since graphics is an ever-expanding facility, the likelihood of follow-on graphics projects, especially within Command Alpha, is high.

- If the team develops a straightforward system, that is, does not attempt to develop reusable components, it will require an initial investment of $40,000. With this type of development process, the new system is projected to produce savings to the command of $20,000 a year for the next 5 years. These savings would mostly result from the reduced paperwork and the reduced amount of man-hours necessary to produce technical drawings.

- If the system is developed by generating reusable components, it is expected to require an initial investment of $65,000. The additional money would be needed to provide more extensive documentation of all phases in the life cycle of the system, to develop more generalized modules for use in the coding process, and to develop the system with a virtual machine approach to hide the details of the hardware. It would also be needed to encapsulate design decisions that are likely to change in the future, to allow more extensive testing, and to create a library and cataloging system for maintaining the reusable information. The savings to the command over the next 5 years are projected to be $35,000 per year. These savings would include reduced costs for future development projects due to the availability of reusable components.

Using these projections, the team leader proceeds to determine which development method to use. He therefore computes the net-present-value coefficients for the two positions, using a discount factor of 12%. Following are his calculations: [Ref. 22].

- For Method I (no reusability):

	Investment	Returns		12%pvf		Present Value
Year 0	$40,000			1.000		
Year 1		$20,000	x	.893	=	$17,860
Year 2		20,000	x	.797	=	15,940
Year 3		20,000	x	.712	=	14,240
Year 4		20,000	x	.636	=	12,720
Year 5		20,000	x	.537	=	10,740
	$40,000					$71,500

Thus, the net present value is $71,500 minus $40,000, or

$31,500. The net-present-value coefficient in this case is given by $31,500/$40,000 and is thus equal to .7875.

- For Method II (with reusability):

	Investment	Returns		12%pvf		Present Value
Year 0	$65,000			1.000		
Year 1		$35,000	x	.893	=	$31,255
Year 2		35,000	x	.797	=	27,895
Year 3		35,000	x	.712	=	24,920
Year 4		35,000	x	.636	=	22,260
Year 5		35,000	x	.537	=	18,795
	$65,000					$125,125

In this case, the net-present-value is $60,125 and the coefficient is $60,125/$65,000, or .925.

The results indicate that Method II will give a higher return on the initial investment. The team leader therefore concludes that it would be advantageous in the long run to expend the capital now to develop reusable components for later use.

For the next step, the team must determine what the requirements of the system will be, that is, what the system should do. There are many questions to consider, such as:

- What menus should be available?
- Should an on-line help facility be provided?
- Should there be a file system to enable the user to save and re-edit at a later time?
- Should predefined shapes be made available?
- Should translation, rotation, and/or modification be available?
- How many different line widths should be made available?
- Should it be possible to include text in a drawing?

- What symbols are important for this command's specific needs?

- What colors, textures, and line styles should be provided?

- Should there be an erase feature?

- Should there be an undo feature to remove only the effects of the last executed command?

- Should there be a fill feature to allow the user to fill in an enclosed area?

- Should freehand drawing be permitted?

- Should different sized text be made available?

The team leader knows that other personnel have previously developed figure generation packages, but he cannot locate any documentation as to how the requirements were determined. Articles on computer graphics relate what is available, but not how these requirements were decided upon. Commercial graphics packages also show what is available, with no information as to how the determination to provide those facilities was made. The team leader concludes, therefore, that his team must evaluate the above questions and formulate their own conclusions. Some gains can be made by ascertaining what is available in other systems, but the determination of the reasons for incorporating some features as opposed to others must be made anew.

Some decisions made in the requirements phase must await the operation of the system to learn whether or not the decision was the best. Unfortunately, except for

critical reviews of some systems, the team can find little or no documentation of what is lacking or insufficient in the operation of similar past systems.

The development team next proceeds to the design of the product. At this point, questions regarding how the system will function must be answered. These include:

- Should the menu(s) remain on the screen at all times?
- Where will the menu(s) be located on the screen?
- How will a selected item be denoted?
- How will straight lines be provided?
- How will line ends be joined?
- How will a file be saved, stored, and retrieved?
- How will help information be provided?
- How will selection be accomplished?
- How will text be entered?
- How will freehand drawing be performed?

The team leader notes that design information is available, however, it is contained in thousands of pages of documentation. Rather than spend the man-hours necessary to determine if any of it can be reused on this project, the team leader decides it will be more cost-effective for the team to proceed on its own in developing the design of the system. He bases this decision on the estimate that the additional man-hours needed to locate possible reusable components from past projects could cost an additional $10,000. Assuming an initial investment of $65,000, this

would represent approximately 13% of the investment, and would account for dubious benefits.

After the design is completed, the software team looks for previously devised test plans that could be reused to verify that the final product meets the requirements. Once again, no data of this sort can be located from prior projects, so the team must produce its own test plans for the system.

The group then proceeds to generation of the code. The team leader determines that the code from previous projects is readily available. Here, he must ascertain whether or not the personnel on the development team will be able to readily understand the previously written code and thereby modify it to meet the needs of their system, or if it will be more advantageous to have the team generate its own code. The team leader concludes that some of the previously generated code has been well documented and is very modular and thus directs the group to use any of these modules, subroutines or functions that can be modified to meet their requirements.

Little operational or maintenance information on past systems can be located by the team. Some data exists in critical reviews found in publications and journals, though, and this data does provide for some reuse. For example, the team learns that users desire an online

prompting so that there is less that must be remembered or looked-up in manuals, so they incorporate this facility.

2. Scenario II.

Command Bravo is a data processing center. It has found that incoming personnel are inexperienced in operating the equipment at hand, and therefore has recently procured computer graphics hardware to devise an interactive training program. A software development team has been formed to produce the graphics package necessary to implement the training program.

The development team leader discovers, through the company representative, that the hardware his command has purchased is quite similar to that purchased by Command Alpha some time ago. He therefore contacts personnel in the development office at Command Alpha to see if any information generated by them can be reused on the present project. Documentation on Command Alpha's figure generation system is sent to Command Bravo.

The Command Bravo team must first select an overall approach to their project. They review the feasibilty documentation of Command Alpha to try to obtain some initial direction for their project. This information proves to be quite useful, especially since the two commands' hardware are so similar. The Command Bravo team discovers that the Alpha personnel reviewed many aspects before selecting an approach. These same issues are

relevant to the current project. From this review, the Bravo personnel decide on a menu-driven, mouse-controlled system.

Next, the team must determine the requirements for the system. Here, there are differences between the two commands since the new system will be used in a different manner, that is, for projecting graphics displays on the computer screen for interactive training simulations. The Bravo development personnel will not themselves be end users of the finished product, as was the case for Command Alpha. Thus, the Bravo team now interacts with the Operations Division personnel to determine what will be most effective capabilities for the new system. A review of the data from the requirements analysis of the Command Alpha group shows that little of this information can be reused since the analysis necessarily concentrated on different aspects.

Proceeding to the design phase of the new system, the Bravo development team determines that some of the previous design information can be reused, although most must be created anew. Data concerning the menus and selection, for example, can be used on the current project. This is due to the fact that the some of the additional funds expended on Command Alpha's project were used to develop generalized menu and selection routines. The team leader determines that reusing this information will provide

a cost savings due to the man-hours that will be reduced on the design development.

Since the new system is being designed to provide different capabilities than the previous system, the testing of the system will be changed. Thus, alternate test plans must be generated. However, unit testing of modules which were developed by reusing Command Alpha modules will be simplified. These modules have basically already been tested in the previously developed system.

The Bravo development team leader notes that very little of the code from the Alpha project will be appropriate for reuse on this system. Modules that implement menu presentation, provide help information, and perform selection can be modified to meet the needs of the new project.

Once again, since the present system is so divergent from the past one, the operational and maintenance information from Command Alpha will basically be inappropriate for reuse.

3. Scenario III.

Command Alpha personnel have been functioning well for some time with their computer graphics equipment and software. They now receive additional, less expensive, graphics hardware to enable them to expand their graphics capabilities throughout the command. The new hardware was competitively procured and is from a different vendor than

"...drawings are the language by which construction personnel communicate." [Ref. 30] We must recognize that the written word has not been an effective "language" by which software personnel have communicated to make reusability more of a reality. We need to develop a dramatically different methodology, therefore, and one that can alleviate some of the fundamental obstacles to reusability.

A software blueprint would be written using standard symbols and notation. It might consist of different types of diagrams to explain various aspects of a software project. For example, there could be a requirements diagram which would describe not only the requirements, but how they they were decided upon. There could be design diagrams to describe both the product and the detailed designs of the system. Then, there could be an integration diagram to show the interfaces, a testing diagram to show the test plans, and a maintenance diagram to describe what has occurred in maintaining the system.

The idea of using diagrams to provide information is supported by the fact that visual aids often assist in our comprehension and thinking processes. "A well-illustrated article is easier to understand and recall because the mind has more material to work with, more external support. Good illustrations help you conjure up your own images and diagrams, making it easier to understand and absorb the

documentation of information produced throughout the life cycle phases of a software development project is crucial to making reusability a reality.

Once a standard form of documentation is developed, we can proceed to the new approach that this thesis proposes, which is the use of software blueprints. The objective of this approach is to develop blueprints, or diagrams, using standardized notation, in which the information generated throughout the development of a software system could be displayed.

Consider the blueprints or diagrams used in architecture or in the development of computer hardware systems. There are floor plans, wiring diagrams, plot plans, circuit diagrams, and more. The advantage of these diagrams is that they present information in a clear, concise manner. This has been a problem area for reusability in software development. We have much information retained from previous projects, but it is not easily retrieved, nor is it understandable. The data is stored in manuals, in volumes of paper, or with automated systems that not everyone can comprehend. Blueprints have the advantage of displaying the necessary information understandably, completely, and in detail. They allow individuals to communicate more effectively about projects because they are able to see what they were discussing in a concise form. Diagrams are used extensively in other professions,

comprehensible to all personnel. The use of standards has long been recognized as vital to other professions:

> Recognized standards are a major aid to all concerned with building construction. They reduce the amount of definitive material to be produced for each project.... Standardization does imply that universally familiar indications, symbols, and terms are more foolproof than untried versions. It requires the consistent use of the same identification for each item in the specifications and on all the drawings. [Ref. 28]

This kind of standardization in architecture, for example, has enabled all personnel involved in the building process to understand the documentation. This, in turn, has allowed them to be able to reuse drawings and thus avoid duplication of effort:

> A productive source of construction know-how for experienced as well as new draftsmen is the office file of completed drawings....Frequently you will be able to find solutions to problems quite similar to the ones you are facing.... [Ref. 29]

The lack of standard documentation in software engineering will continue to make reusability difficult. There are problems noted today with the systems that are in use which attempt to document certain life cycle phases, most notably, the requirements and design phases. Consider, for example, the Structure Analysis and Design Technique (SADT) and the Software Requirements Engineering Methodology (SREM), which were previously discussed. These systems, even on a local level, have not gained widespread usage due to difficulty in understanding them and the extensive training needed to do so. Thus, the need to develop a standard notation for the

reusability in mind. We must plan for additional development time and effort, and, thus, additional development cost, to make reusable components available in the future. Hence, we must consider the need to spend additional dollars today for savings tomorrow.

It is evident that software engineering can benefit by applying some of the methods that have been long practiced and have been of great use to other engineering fields. We need to devote our efforts toward making the construction of software systems an effective engineering discipline.

B. PROPOSED NEW APPROACH

Software reusability will always remain a basically local effort unless we make a concerted effort to develop a new methodology to provide a support system for generating the reuse of software components. The first step is to incorporate the best concepts from the more experienced engineering fields. Then we must expand these to find the most advantageous approach for software production. This thesis now proposes ideas for that approach.

For reusability to gain widespread adoption, standards must first be developed. One of the main stumbling blocks to reuse of software components has been a lack of readability, and, therefore, understandability, of previously developed components. There are no standards to follow which would make the documentation more

work. Additionally, the drawings for a specific project are saved and are then available for reuse where appropriate.

Also, prior to the actual erection of a building, a great deal of time and effort is expended in preparing the necessary drawings or documents to gather together all the information required to build the structure [Ref. 26]. A similar approach is not always taken in the field of software construction. The problem was succinctly stated by Yaohan Chu:

> In engineering methodology, engineers build by a plan, whether the product is intended for mass production or for one-time use only. First, the engineer develops the design; then, the product is constructed. In addition, a design document--the engineering blueprint--is produced....This long-practiced methodology contrasts with the current practice of programmers, who attempt to write a program without a comprehensive design and (equally important) an understandable, concise design document. [Ref. 27]

Often, only minimal information regarding the requirements, design, and testing of a system is collected before actual coding is begun. The emphasis is placed on meeting the schedule for having the system available. This is especially evident when work is being done in-house, where it is sometimes felt that the greatest need is to get the system operational, with additions or corrections easily incorporated later. It is this type of thinking that provides an obstacle to reusability. If specific effort is not taken with the development aspects of a software project, the system cannot hope to be produced with future

V. NEW APPROACH

A. SOFTWARE ENGINEERING CONCEPT

The American Heritage Dictionary describes engineering as "the application of scientific principles to practical ends as the design, construction, and operation of efficient and economical structures, equipment, and systems." [Ref. 25] In this regard, we consider the design, construction, and operation of computer software systems to constitute an engineering discipline. Correspondingly, definitions of software engineering have been presented, such as the following by Barry Boehm: "Software Engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation." [Ref. 26] Thus, we speak of "software engineering" and "software engineers" when discussing the development of computer software systems. However, we do not approach the task with the same types of tools, or with the same conviction, that other engineering fields exhibit.

Consider, for example, the field of architecture, involving the design and construction of buildings. In practice, for this profession, there exist blueprints, floor plans, wiring diagrams, standard symbols, and so forth. These tools are produced in a standard way, so that personnel in the same profession can understand another's

by using those components, reusability may create more of a loss than a gain. Also to be considered in a cost/benefit study is whether to expend additional capital now to develop reusable components for future use. This is a difficult aspect to contemplate on anything but a local level at the present time, since most software managers must consider their own projects foremost in order to be considered effective and efficient at their jobs. This type of analysis should be actively pursued within development offices, though, to generate as much reuse of software as possible.

2. Determine if any information that could be reused on the project at hand exists. The type of system being developed may be so new that no previous project can provide useful information. In such a situation, extra care should be taken to document all efforts on the current project for the benefit of future applications.

3. Consider the maturity of the type of system being developed. A more mature system will be able to provide not only more, but more reliable, reusable components.

task." [Ref. 25] This type of information should be well documented and readily available to enable future development personnel to make the best, most informed decisions.

The basic maturity of the type of system being developed must be considered when contemplating reusability. Scenario I illustrates a situation in which the type of system being developed is basically new. In such a case, very little information can be found and, thus, little will be available for reuse. As a system matures, more information regarding it becomes available, for each life cycle phase, and more components become suitable for reuse.

E. GUIDELINES

The scenarios presented and the above discussion suggest certain aspects which should be considered in performing a trade-off analysis to determine if reuse of software components is applicable to a certain development project. These are outlined below as possible guidelines for the software development manager:

1. Perform a cost/benefit analysis. This should be a computational analysis based on the best possible estimates of the values of costs and benefits. Since the economic issue of reusability is the most important one, this type of analysis is crucial for any trade-off decision. It includes several aspects. First, one must consider the expected man-hours of effort necessary to locate reusable components versus the expected man-hours needed to develop the project from scratch. Additionally, if incorporating reusable components will generate more work in modifying the components to meet the user's need than will be saved

example, benefits recognized in the maintenance phase must be discounted to account for the time difference.

Additionally, as Scenario I depicts, it may not be cost effective to attempt to locate reusable components from prior projects. If more man-hours would be expended to discern the necessary reusable information from massive volumes of documentation than would be to develop the project without the reusable information, then trying to incorporate reusability is not cost effective. According to Wagner and LaHood, "...unpublished materials...those in files...represent the project documentation of about 15 software design projects concerned with computer graphics, and contain somewhere between 10,000 and 20,000 pages." [Ref. 24] Thus, for a situation such as related in Scenario I, it would not be cost-effective to try to find reusable data.

It is also important to consider all information about a software project as available for reuse at any time during the development of a new system. Operational or maintenance information may have a bearing on the requirements or design of future systems. As Foley and Van Dam pointed out, "A classic study demonstrated 100% differences in speed and 200% differences in error rates among several techniques for picking displayed words. Two different interactive graphics drafting systems, designed to do the same job, have shown differences of 100% in the overall time to complete a given

so much of the art of programming, very little of it is recorded in the published literature." [Ref. 23]

Also to be considered is whether to develop a system for future reusability. To make this determination, as demonstrated in the first scenario, a cost/benefit analysis must be performed. This is most effectively done by computing the net-present-value coefficients for the situations being compared. First, the potential costs and benefits must be identified, and values assigned to these. Then, because the value of money changes over time, the future values are discounted backward in time by multiplying them by a discount factor, or present value factor (pvf). This net-present-value (NPV) technique, therefore, discounts the costs and benefits to the present year and compares them, by way of a NPV coefficient. The coefficient is the ratio between the net-present-value and the initial investment. These calculations provide an effective means of determining how to proceed with a project.

There are other factors to keep in mind in a trade-off analysis. An important issue is whether one can anticipate follow-on projects which will benefit from reusable components. If the indication is that the system being developed is likely to be a single effort, there is no point in developing the system by designing reusable components.

It is also important to identify not only what the costs and benefits are, but where in the life cycle they are. For

capabilities. Here, some reuse may be warranted. It also presents a situation in which a decision for future reuse is not made. More reusability can be seen with a situation such as Scenario III illustrates. In this instance, the hardware is different but the requirements for the software are basically the same. The most reuse of software components can be seen with an example such as Scenario IV presents, in which both hardware and software are generally the same as a past developed system. Much more information could be reused in a case such as this if the original system were developed for future reusability.

Scenario I also raises other points about the issue of reusability. Even if other similar systems exist elsewhere, that is, external to the command, software components are only effectively available for reuse if they have been well documented and are easily retrievable. Thus, if a development team cannot locate documentation on a particular phase for a project, they should ensure that their efforts are well recorded for the benefit of future projects. This should include all information, such as ideas, reasons for decisions, and the decisions themselves. This factor has been found to be the biggest deterent to using reusable software components, that is, little or nothing is documented from past experiences. As Wagner and LaHood noted in discussing the design of computer graphics, "Like

must be regenerated for a complete understanding of the project. The one aspect that allows reuse of Command Bravo's work is that the systems are so similar.

Since this system is to be used onboard ships, there will be some differences. The requirements must be changed to reflect the need for increased system reliability and to provide for the new information which is to be used in the training program. The team leader decides that the previously devised test plans will be appropriate, but must, however, be expanded to more rigorously test the system for shipboard use.

The development team recognizes that, as a shipboard system, this software can expect to remain in use for some time. Hence, more emphasis is placed on developing an extremely reliable and effective system.

D. DISCUSSION

The above scenarios describe several situations in which varying degrees of reusability are appropriate. Scenario I is indicative of a situation in which the hardware and the software for the system being developed are new. In such a case, little from past projects can be reused as the systems are so dissimilar. This scenario also describes the situation in which an explicit decision is made to develop a system for later reuse of components. Scenario II presents the case for similar hardware with differing software

changes would be warranted for the new system. Using this data, they discover some minor design changes that will effectuate improvements in the speed of the system.

4. Scenario IV.

Command Charlie has been tasked with developing an interactive computer graphics system for shipboard use. This system is to be used to train personnel in the use of various shipboard equipment. The development team leader finds that the contract for the computer hardware has been won by the same company which provided graphics hardware to Command Bravo. He also notes the similarities in the requirements of the systems for both commands, and thus contacts personnel at Command Bravo to determine if any of their development information can be reused.

The development team determines that some of Command Bravo's data can be reused since the hardware and software for the new system will be basically the same as the previous one. The overall approach will be generally identical, as will much of the requirements. The design of the system will also be similar, and much of the code can be modified to meet the new demands. The Charlie team becomes aware, however, that the Bravo personnel did not develop their system with future reusability in mind. Thus, much of the documentation which would have greatly assisted the Charlie team does not exist. Some duplication of effort results since many of the ideas considered by the Bravo team

the previous equipment. A graphics figure generation system must also be devised for this hardware as none was provided with the equipment. A development team is formed to accomplish this task, consisting of different personnel than those who originally developed a graphics package for the command.

The development team leader notes that the present project is exactly like the previous project, except for some variations due to the differences in the hardware. Therefore, the team retrieves all the documentation from the past project. Since the command desires to retain the features of the system at hand, the development team determines that it will be able to reuse much of the information previously generated.

Very little effort must be expended on determining the approach. It will be the same as that of the previous project. Additionally, the requirements will be identical, as will be the test plans. Since the original system was developed with future reuse in mind, the team leader determines that it will be very cost-effective to use the previously designed components. He estimates that developing a system from scratch would require an investment of $40,000 and developing the system with reusable components will generate a reduction of $20,000.

The development team now looks at operational and maintenance information of the older system to decide if any

material." [Ref. 31] Consider the following description of a triple-bus microprocessor architecture:

> Two input-buses named A-bus and B-bus are provided. The A-bus is connected to the right input of the ALU, and the B-bus is connected to the left input of the ALU....Also, results can be gated on the D-bus independently of the two source-buses. If the result must be written back to one of the source registers, a buffering is required. This buffering must be provided on the D-bus or directly within the registers.... [Ref. 32]

This description becomes much easier to comprehend by reading the words and viewing the illustration given on the next page [Ref. 33].

If we can, therefore, develop an effective notation and blueprint scheme for describing the construction of software systems, we will have taken the first definitive step toward supporting reusability. Personnel will be able to more easily retrieve data from past projects, and it will be provided in a form that all software engineers can understand.

The task of developing a software blueprint and notation will not be an easy one and should not be underestimated. To be effective, the blueprint must contain large amounts of information about each of the various life cycle phases. Ways of presenting procedural information, data structures, interface information, and data concerning other development issues, must be devised. Thus, there is a need for a standard notation to enable us to present information by symbols, and in a way that will be understandable to all

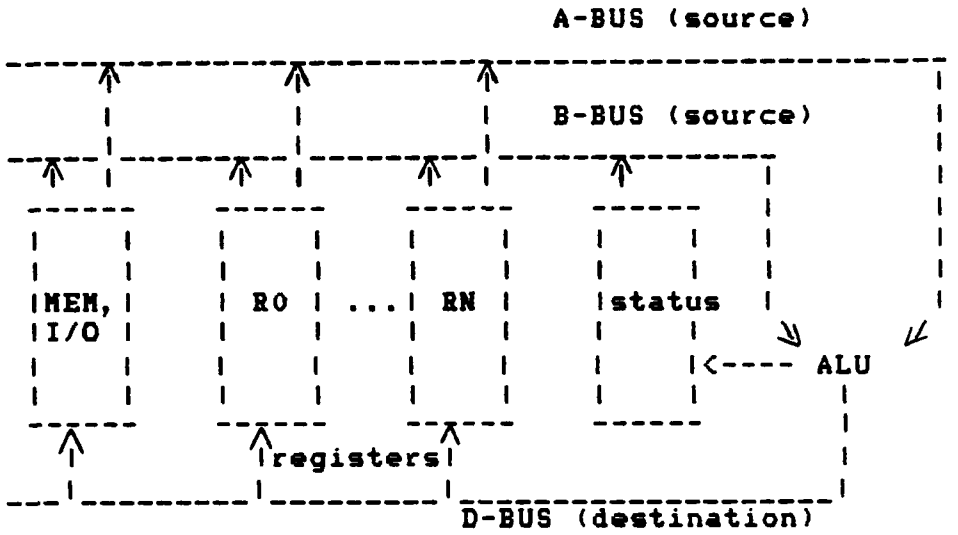


Figure 3. Triple-Bus Architecture

software personnel. We must also consider the difficult questions of how to process the information that would be available in a software blueprint and how to retrieve that information. Thus, developing a reusable software support system will be an arduous and complex process, but a necessary one if widespread reusabilty is ever to become a reality.

The current software crisis makes it imperative to become inventive, to not become mired in old techniques that have not supported reusability in the past, and cannot be expected to do so in the future. Logically, a good place to start to find a plausible approach is with techniques that are effective in other engineering disciplines, and techniques, such as standardization, that have proven to be responsive throughout history.

The basic idea of a drawing to present information is not new. It has been attempted in software development with flow charts, HIPO charts, SADT diagrams, and so forth. What is new is the idea of standardizing the presentation of the information, and using diagrams to represent as much of the life cycle documentation as possible. Both of these concepts have been successful in the past in other areas and may provide the answer to increasing the availability of reusable software components.

VI. CONCLUSIONS AND RECOMMENDATIONS

As this thesis suggests, the incorporation of reusability into software development projects may not currently be a practical consideration. There is a danger in assuming that reusability is of definite economic advantage and should, therefore, be pursued in all cases. The tendency in most articles on the topic, up to this point, has been to embrace the concept of the reuse of software as a cure for the software crisis. In fact, using present methodologies, it may have the opposite effect. That is to say, by attempting to locate reusable components we may actually expend more time, effort, and cost than we would by producing the product from scratch. There are many factors to consider--cost, benefits, human factors, the maturity of the type of system being developed, and the availability of reusable components. Thus, it is imperative, with present methodologies, that an effective trade-off analysis be undertaken to determine if reusability is advisable. It is important to recognize that it may not be.

We can see that although reusability was first discussed in 1968, not a great deal of progress has been made. True, we perceive that the reuse of software components would be valuable in alleviating the software crisis. However, we have not made great strides toward that end. Inasmuch as we

cannot find widespread application of reusability, we must look toward developing new approaches for advancing its use. This thesis suggests that we adopt some effective tools from other engineering disciplines, namely, standardization and blueprints, and forge ahead. Too many years have been wasted in only discussing how great the concept of reusability is, without actively and vigorously trying to make it a reality on a wide scale.

We must now seek to get the project underway. This could be accomplished by setting up a specific project for developing reusable software. Some form of standard notation should be developed, as well as the format for software blueprints. The initial development of components, which are currently frequently produced in separate projects, into a reusable form should be undertaken. A library and cataloging system must also be devised so that development personnel will be able to rapidly locate reusable components.

The only way reusable software is going to be a reality on anything but a local level, is by developing a strong support system for it. Also important is the need to educate personnel on the new methodologies, once they are completed. The system should be documented, personnel should be trained, and managers should insist upon its use.

Until such measures are taken, however, we must be realize that trade-offs are involved, and attempt to make

the most informed decisions about reusability. We should try to incorporate reusable components and develop them, whenever possible, at the local level. In that way, even with the drawbacks of today's techniques, we will be able to achieve as much benefit as possible from reusability.

LIST OF REFERENCES

1. Joint Logistics Commanders Joint Policy Corrdinating Group on Computer Resource Management, "Report of the Panel on Software Reusability," Proceedings of the Software Workshop, 1 November 1981.

2. Workshop on Reusability in Programming, ITT, September 1983.

3. IEEE Transactions on Software Engineering, Vol. SE-10, No. 5, September 1984.

4. Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management, p. 3.

5. Ibid, p. 2.

6. Boehm, Barry W., Software Engineering Economics, p. 36, Prentice-Hall, Inc., 1981.

7. Standish, Thomas A., "Software Reuse," Workshop on Reusability in Programming, ITT, p. 45, September 1983.

8. Leavitt, Don, "Reusable Code Chops 60% Off Creation of Business Programs," Computerworld, Vol. 13, p. 1, 29 October 1979.

9. Boehm, p. 18.

10. Freeman, Peter, "Requirements Analysis and Specification: The First Step," Tutorial on Software Design Techniques, 4th ed., pp.80-1, 1983.

11. Ibid, pp. 82-3.

12. Lanergan, Robert G. and Dugan, Denis K., "Software Engineering with Reusable Designs and Code," Association for Computing Machinery Proceedings of the Annual Conference, p. 296, Fall 1981.

13. Boehm, Barry W., "Software Engineering," IEEE Trans. Computers, pp. 1226-41, December 1976.

14. Horowitz, Ellis, and Munson, John B., "An Expansive View of Reusable Software," Workshop on Reusability in Programming, ITT, p. 254, September 1983.

15. Ibid, p. 250.

16. Bruton, Eric, The History of Clocks and Watches, p. 145, Crescent Books, 1979.

17. Wegner, Peter, "Reflections on Capital-Intensive Software Technology," Software Engineering Notes, Vol. 7, p. 24, October 1982.

18. Ibid, p. 25.

19. Cooper, John D., CDR, USN, "Computers in Tactical Systems," Computers in the Navy, p. 122, 1976.

20. Naval Shore Electronics Criteria, Digital Computer Systems, NAVELEX 0101,111, Vol. I, p. 3-7, March 1972.

21. Perry, Oliver H., III, LT, USN, and Fox, Richard J., LT, USN, "Computers in Naval Fire Control Systems," Computers in the Navy, p. 165, 1976.

22. Powers, M. J., Adams, D. R., Mills, H. D., Computer Information Systems Development: Analysis and Design, pp. 184-208, South-Western Publishing Co., 1984.

23. Wagner, F. V., and LaHood, J., "Computer Graphics," Computer Graphics, Utility/Production/Art, Thompson Book Company, 1967.

24. Ibid.

25. Foley, J. D., and Van Dam, A., Fundamentals of Computer Graphics, pp. 222-42, Addison-Wesley Publishing Company, 1984.

26. The American Heritage Dictionary of the English Language, p. 239, Dell Publishing Co., Inc., 1972.

27. Boehm, p. 16.

28. Chu, Yoahan, Software Blueprint and Examples, p. 1, Lexington Books, 1982.

29. Hooper, Lee, Introduction to Construction Drafting, p. 125, Prentice-Hall, Inc., 1971.

30. Ibid, p. 229.

31. Huth, Mark W., Basic Construction Blueprint Reading, p. 39, Delmar Publishers, 1980.

32. Benzon, Bill, "The Visual Mind and the Macintosh," Byte, Vol. 10, No. 1, p. 120, January 1985.

33. Zaks, Rodnay, From Chips to Systems: An Introduction to Microprocessors, p. 49, Sybex, 1981.

34. Ibid, p. 49.

INITIAL DISTRIBUTION LIST

		No. copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Prof. Gordon H. Bradley, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	4
4.	Prof. Bruce J. MacLennan, Code 52Ml Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	1
5.	CAPT Crandall Naval Sea Systems Command (SEA 61Y) Department of the Navy Washington, DC 20362-5101	1
6.	LT Cynthia A. Murnan 232-B North Spruce St. Batavia, NY 14020	2
7.	Computer Technologies Curricular Office Code 37 Naval Postgraduate School Monterey, California 93943-5100	1

END

FILMED

11-85

DTIC